Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

3D игра в жанре Action-Adventure на движке Unity

Пилецкий Олег Антонович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к.т.н. Литвинов Юрий Викторович

Санкт-Петербург
2023

Оглавление

Введение

Игровая индустрия — одна из самых обширных областей индустрии развлечений, в которой задействованы десятки тысяч человек по всему миру. Для их разработки используются различные игровые движки, и одним из самых популярных и несложных в освоении является Unity Engine.

Видеоигры традиционно делятся на множество жанров, и одним из самых известных является приключенческий экшен, в котором сочетаются как элементы квестовых игр, такие как наличие загадок и сюжета, так и основные особенности экшен-игр: необходимость быстро реагировать и координировать свои действия. У игр этого жанра достаточно много любителей, что подтверждается объемом их продаж.

- Death Stranding — 5 миллионов[1]
- Hollow Knight — около 3 миллионов[2]
- Supraland — 600 тысяч[3]

Чтобы изучить движок Unity и получить опыт разработки игр, было принято решение создать игру в этом жанре, основываясь на идеях из таких игр, как The Legend of Zelda, Deep Rock Galactic и Outer Wilds. Проект планируется в первую очередь учебным, поэтому вначале предстоит ознакомиться с обучающими материалами. Стоит заметить, что в игре как в продукте предполагается наличие множества различных возможностей взаимодействовать с миром, которые необходимо реализовать разработчику так, чтобы не только предоставить игроку свободу действий, но и избежать его непредсказуемого поведения.

Данная игра будет представлять собой шутер от первого лица с механиками для перемещения, такими как притягивающий крюк, которые будут использоваться игроком в постепенно открывающемся мире. Целью снабженного оружием игрока является исследование и заполнение

[1] Death Stranding has sold five million copies, дата обращения: 13.12.2022
[2] Hollow Knight Silksong Revealed, дата обращения: 13.12.2022
[3] 600k+ players, дата обращения: 13.12.2022

журнала на другой планете, на которую отправился главный герой в составе исследовательской миссии и на которой он может в том числе найти поселения других людей.

1. Постановка задачи

Целью работы является изучение игрового движка Unity путем создания прототипа игры в жанре “приключенческий экшен от первого лица” и получения отзывов о ней. Для ее достижения были поставлены следующие задачи.

- Провести обзор существующих открытых аналогов и руководств по разработке игр на Unity на известных ресурсах
- Разработать способы взаимодействия игрока с миром
  - Передвижение
  - Несколько видов оружия (пистолет, дробовик, автомат, притягивающий крюк)
  - Пользовательский интерфейс, отображающий его характеристики
- Создать систему сохранения игрового прогресса, состояния мира игры и пользовательских настроек
- Создать мир с открывающимися по ходу игры локациями, с обычными и несколькими особыми врагами, а также нейтральными персонажами
- Создать внутриигровую энциклопедию снаряжения и неигровых персонажей
- Провести апробацию среди пользователей

2. Обзор

В данном разделе приведены обзор некоторых игр с открытым исходным кодом и обзор технологий, используемых при разработке игры.

2.1. Открытые аналоги

Было решено выбрать несколько игр, созданных для обучения компанией Unity Technologies, и разобрать их.

- FPS Microgame доступен для скачивания из Unity Hub. В нем присутствует небольшой тестовый уровень с игровым персонажем, у которого есть пистолет-пулемет и реактивный ранец, и двумя врагами. Также он включает в себя небольшие интерактивные руководства по добавлению готовых врагов или созданию уровней из готовых частей, что было полезно, чтобы узнать основы редактора. Архитектурно же этот проект сложно расширяем, но есть возможность взять из него некоторые системы подбора предметов и систему здоровья, разделенную на акторов, само здоровье и объекты, способные получать урон.

- Adventure Sample Game [6] представляет собой прототип Point-and-Click приключенческой игры с видом сверху с двумя небольшими уровнями и несколькими интерактивными предметами. Целью игрока в нем является нахождение нескольких предметов, чтобы замаскироваться под другого человека и пройти контроль. Архитектурно в нем содержатся легко расширяемые системы инвентаря, взаимодействия с окружающими предметами, реакций на события, сохранения, реализованные при помощи наследников класса `ScriptableObject`.

Эти прототипы помогают разобраться с работой движка Unity и дают общее представление о том, как составлять архитектуру проектов.

2.2. Используемые технологии

Работа над игрой ведется на движке Unity версии 2021.3, скрипты пишутся на языке C# 9.0. Также из официального магазина ассетов были взяты пакеты

- Input System, позволяющий абстрагировать системы ввода и выполняемые команды
- Пакеты со спрайтами и звуками

Основными классами в Unity Engine, от которых наследуется большинство других, являются `MonoBehaviour` [3], отвечающий за поведение объектов на игровой сцене, и `ScriptableObject` [5], позволяющий хранить информацию независимо от экземпляров объектов на сцене. Чтобы их использовать, нужно перенести их в специальное окно, называемое инспектором и показывающее информацию о выбранном объекте на сцене. Каждому такому объекту соответствует компонент `Transform`, хранящий информацию о местоположении, повороте и размере.

Для создания событий, которые можно модифицировать в инспекторе, используется класс `UnityEvent`, а также абстракция `GameEvent` и `GameEventListener` [1], позволяющая отделить вызываемое событие от реакции на него. Принцип работы показан на рисунке 1.

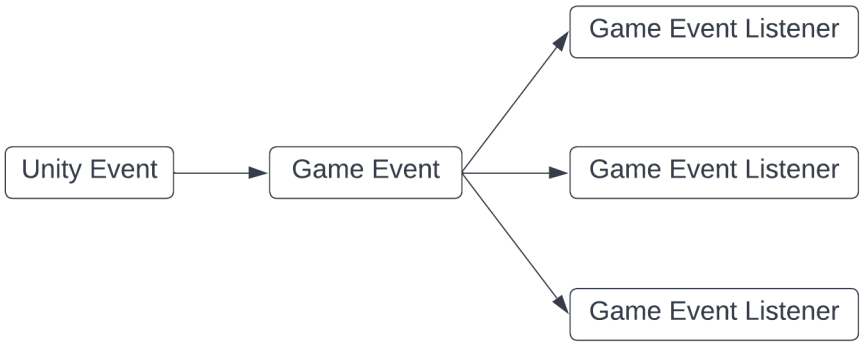

Рис. 1: Система событий

Для работы с физикой персонажа был использован компонент `Rigidbody`, симулирующий твердое тело. Для выполнения повторяющихся действий

через равные интервалы времени использовались специальные итераторы, называемые корутинами [2]. Также важным компонентом для отрисовки пользовательского интерфейса является холст (Canvas), на котором содержатся остальные компоненты:

- Слайдер, визуально отображающий значения от 0 до 1
- Текст, использующий пакет TextMeshPro
- Layout Group, служащий разметкой для дочерних элементов интерфейса

3. Реализация

В первом семестре работа велась в основном над игровым персонажем и возможностями его взаимодействия с игровым миром. Глобально его реализацию можно разделить на 3 отдельных части: передвижение, управление оружием, управление инвентарем. Игрок для доступа ко всем ним использует специальный интерфейс и заданные кнопки на клавиатуре.

Кроме того, была создана система сохранения игрового прогресса и настроек в файл формата JSON.

При открытии игры появляется меню, в котором можно начать новую игру, продолжить уже начатую, открыть меню настроек или закрыть приложение.

3.1. Управление игровым персонажем

Для обработки ввода был использован пакет Input System, позволяющий связать требуемые способы управления и связанные с ними действия в отдельном окне внутри Unity. Для обработки ввода был создан класс `InputController`, в котором происходит получение ввода и перенаправление его на нужные методы, лежащие внутри других классов-компонентов `MonoBehaviour` на игроке. В скрипте `PlayerMovement` используется метод `AddForce` компонента `Rigidbody`, принимающий вектор силы и режим применения силы, в данном случае — изменение скорости. Чтобы скорость не набиралась бесконтрольно, используется метод `LimitVelocity`, ограничивающий ее. При беге эта максимальная скорость повышается.

Для поворота камеры был создан скрипт `PlayerLook`, использующий компонент `Transform`, присущий всем объектам на сцене и содержащий поворот, позицию и размер объекта. Архитектура системы управления персонажем представлена на рисунке 2.

Для прыжка необходимо, чтобы персонаж находился на поверхности, наклон которой не превышает 45 градусов. Для проверки этого используется статический метод `Physics.SphereCast` [4], который про-

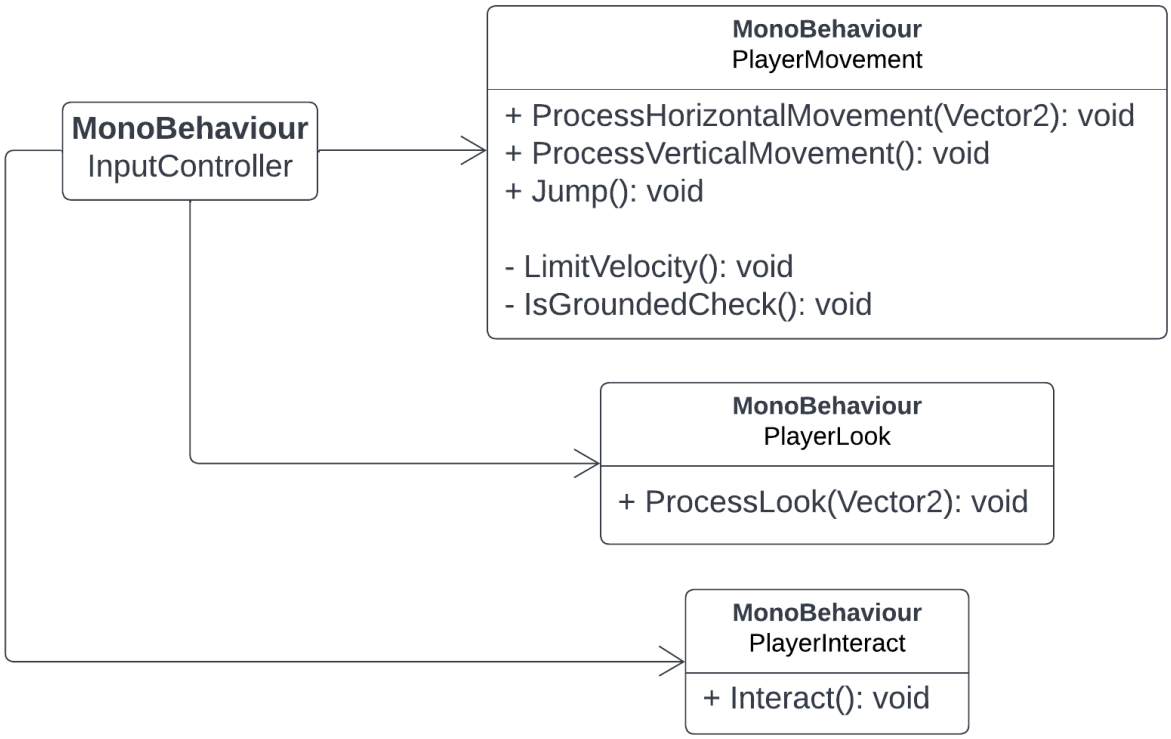

Рис. 2: Система управления игровым персонажем

водит сферу определенного размера в указанном направлении. Если она касается поверхности, проверяется угол, и если он меньше указанного значения, то персонаж может прыгать. Эта проверка проводится каждый кадр, так как она также используется, чтобы применять трение к персонажу и чтобы персонаж не менял бег на ходьбу или обратно во время прыжка.

3.2. Система оружия

Были реализованы 3 типа оружий — автоматический пистолет-пулемет и неавтоматические пистолет и крюк-кошка. Классы `Gun` и `GrapplingGun` реализуют интерфейс `IWeapon`, позволяющий стрелять и перезаряжаться, и наследуются от `MonoBehaviour`, что позволяет повесить их на объекты оружий. Первый при стрельбе инстанцирует пули, летящие по прямой определенное количество времени, а второй при помощи компонента `LineRenderer` инстанцирует линию определенной длины в ука-

занном направлении и, если она столкнулась с объектом на расстоянии меньше максимального, игрок начинает притягиваться к нему.

Архитектура системы оружий представлена на рисунке 3.

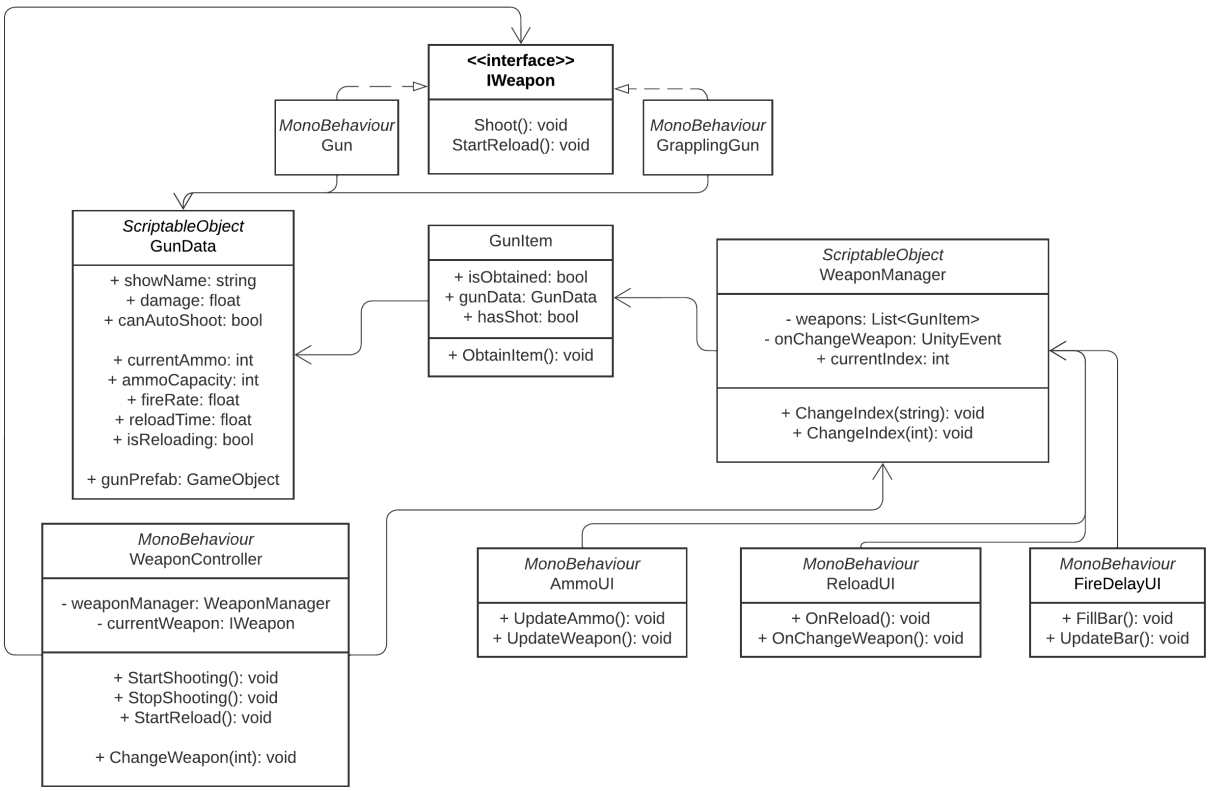

Рис. 3: Система оружий

Информация о характеристиках каждого отдельного вида вооружения хранится в классе `GunData`, который наследуется от `ScriptableObject`, а именно название, урон, текущее количество патронов и сколько патронов вмещает магазин, скорость стрельбы и время перезарядки, а также ”префаб” объекта, то есть готовый объект, который можно поместить на игровую сцену.

Класс `WeaponManager`, также наследующийся от `ScriptableObject` и управляющий информацией обо всех оружиях на игроке, оперирует классами `GunItem`, содержащими характеристики, а также информацию, меняющуюся во время игрового процесса. Чтобы знать, где спавнить модель оружия, используется ссылка на дочерний пустой объект `WeaponHolder`. Прорисовка оружий происходит в классе `WeaponController`, наследующимся от `MonoBehaviour` и находящимся на игроке. Он, в свою

очередь, получает команды стрельбы или перезарядки из `InputController` и использует скрипты, навешенные на оружия и реализующие интерфейс `IWeapon`, позволяющий стрелять и перезаряжаться. Для смены оружия используется метод `ChangeWeapon`, принимающий его номер. Он меняет текущий индекс в менеджере и инстанцирует у игрока необходимое оружие.

Для отображения патронов, задержки между выстрелами и перезарядки использовался элемент холст (Canvas), позволяющий настраивать и отрисовывать пользовательский интерфейс. Для этого были созданы следующие элементы.

- **AmmoUI**. Он отображает текущее и максимальное количество патронов. Для этого были использованы 3 компонента Game Event Listener, слушающие события `OnAmmoChanged`, `OnReloadFinished` и `OnWeaponChanged`. Первые два вызывают метод, обновляющий текст, который отображает текущее количество патронов в магазине, третий — метод, обновляющий текст и с текущим, и с максимальным количество патронов, но уже на другом оружии. В самом объекте также содержится ссылка на Weapon Manager, позволяющая сразу получать нужную информацию при активации событий.

- **FireDelayUI**. Он активен только при взятии игроком неавтоматических оружий, слушающий те же события. Для индикации оставшейся задержки был использован компонент `Slider`. При выстреле он обнуляется и с помощью корутины `FillBarByTime` заполняется за необходимое время.

- **ReloadUI**. Он появляется при перезарядке и визуально отображающий оставшееся время до ее окончания при помощи слайдера. Он слушает события `OnReloadStarted` и `OnChangeWeapon`. Первый вызывает корутину `FillReloadBar`, которая заполняет индикатор перезарядки до максимального значения, пока оружие перезаряжается, а второй устанавливает значение индикатора равным единице.

Для непрерывной стрельбы используются корутины: при удерживании кнопки выстрела запускается корутина, которая вызывает метод `gunData.fireRate` раз в секунду. Для задержки используется инструкция `yield return fireDelay`, где `fireDelay` — кэшированный объект типа `WaitForSeconds`. При отпускании кнопки эта корутина останавливается.

Для одиночной стрельбы также используется корутина, но она устанавливает булевое значение `HasShot` в объекте `GunItem`, соответствующем текущему оружию, в true, и так же вызывает ожидание при помощи `yield return fireDelay`, после которого это значение переходит обратно в `false`. При смене оружия это состояние сохраняется.

3.3. Система инвентаря

Информация об отдельном предмете: название, спрайт, может ли персонаж взять предмет в руки и максимальное число этого предмета — содержится в классе `InventoryItemData`, наследующемся от `ScriptableObject`.

За сам инвентарь отвечает класс `InventoryManager`, также наследующийся от него. Он оперирует сериализуемыми классами `InventoryItem`, которые содержат информацию о предмете, а также его количество в инвентаре. Внутри менеджер содержит список всех предметов, а также словарь для быстрого доступа, где ключ — это информация о предмете, а значение — сам предмет.

Для того, чтобы подобрать предмет с земли, на него навешивается компонент `InventoryPickup`, реализующий интерфейс `IInteractable` с одним методом — `Interact`. Он содержит в себе ссылку на менеджер инвентаря и на скрипт с информацией об объекте. При взаимодействии с ним проверяется, не содержится ли в инвентаре по ссылке максимум предметов, если нет, то он добавляется в инвентарь и удаляется со сцены. В ином случае ничего не происходит.

Интерфейс инвентаря выполнен при помощи объекта `Inventory` на том же холсте, что и интерфейс для оружия. На нем есть скрипт `InventoryUI`,

ответственный за его отображение. В качестве отображения предметов используется префаб `ItemSlot`, показывающий спрайт предмета, его название и количество. Внутри него содержится также ссылка на объект `InventoryItemData`, откуда и берется информация. Если на предмете стоит флажок `canHandle`, то при нажатии на него предмет возьмется в руки и его можно будет использовать. В данный момент это сделано при помощи передачи в `WeaponManager` названия предмета и смены текущего индекса оружия на необходимый.

Архитектура системы инвентаря представлена на рисунке 4.

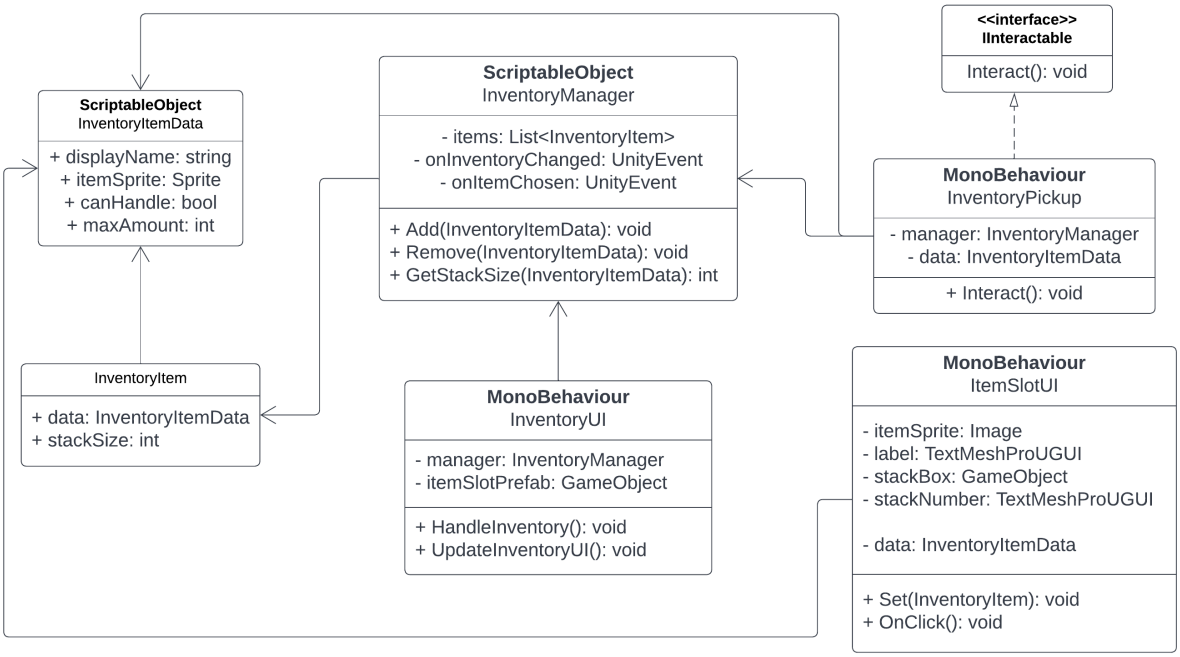

Рис. 4: Система инвентаря

Дочерним объектом инвентаря является `BigInventory`, который становится активным после нажатия на кнопку TAB и показывает на холсте содержимое инвентаря, при этом игрок лишается возможности стрелять или передвигаться. При повторном нажатии на нее, либо при нажатии мышью по кнопке в углу инвентаря, он закрывается, и игрок снова может выполнять эти действия.

3.4. Система сохранения

Архитектура системы сохранения и загрузки игровых данных представлена на рисунке 5.

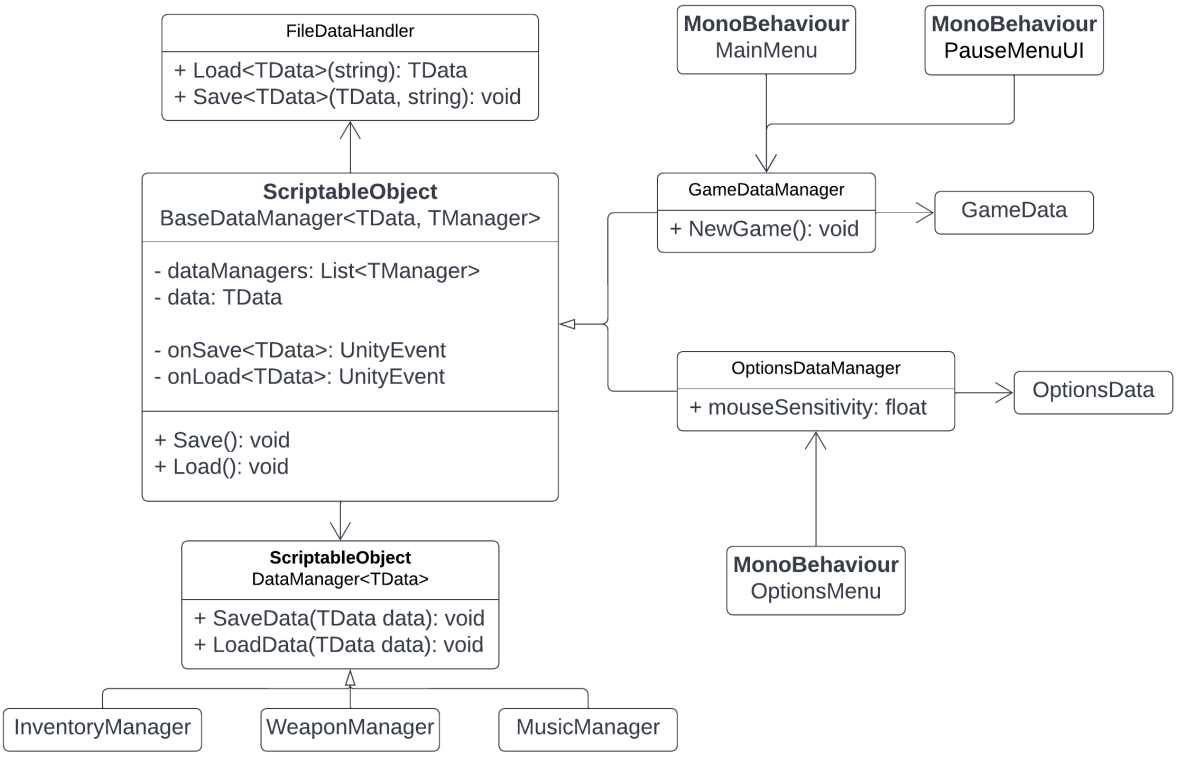

Рис. 5: Система сохранения

Для управления сохраняемыми данными был создан абстрактный класс `BaseDataManager`, от которого наследуются `OptionsDataManager`, отвечающий за настройки игры, и `GameDataManager`, отвечающий за информацию об игровом прогрессе и игроке.

Игровые данные хранятся в сериализуемом классе `GameData`, который управляется через `GameDataManager`. Он хранит в себе список из тех классов, которые управляют данными, то есть `InventoryManager` и `WeaponManager`, которые наследуются от абстрактного класса `DataManager` с типом `GameData`. При сохранении у них вызываются методы, сохраняющие информацию в `GameData` и в файл формата JSON при помощи статического класса `FileDataHandler`. При загрузке эта информация оттуда выгружается. Чтобы сохранить специфичную для игрового процесса информацию, вызываются события `OnSaveGame` и `OnLoadGame`

с типом GameData, чьими слушателями являются компоненты, информация о которых должна быть сохранена или загружена. В самих компонентах вызываются методы OnSave или OnLoad, получающие в качестве параметра ссылку на объект типа GameData и получающие доступ к необходимым параметрам.

Данные настроек хранятся в файле OptionsData, управляемом при помощи OptionsDataManager. Он также хранит в себе список классов, управляющих неизменяемыми данными, в данном случае — MusicManager, наследующегося от абстрактного класса DataManager с типом OptionsData. Он работает по тому же принципу. Для сохранения чувствительности мыши используются события OnSaveOptions и OnLoadOptions с типом OptionsData, слушателем является игровой персонаж, который вызывает метод Save или Load у компонента PlayerLook, отвечающего за поворот камеры. Сами настройки можно поменять в главном меню.

Пользователь может сохраниться в специальном меню паузы, при открытии которого игровое время и вся физика внутри игры останавливаются. Это сделано при помощи установки глобального параметра timeScale класса Time равным нулю.

Заключение

В течение семестра были достигнуты следующие результаты:

- Проведен обзор существующих аналогов и руководств по разработке игр на Unity
- Разработаны передвижение игрока, система оружия с тремя их видами, инвентарь, пользовательский интерфейс
- Создана система сохранения игрового прогресса, состояния мира игры и пользовательских настроек

В следующем семестре планируется

- Создать игровой мир
- Создать неигровых персонажей
- Создать внутриигровую энциклопедию
- Провести апробацию среди пользователей

Исходный код проекта доступен на сервисе GitHub[4].

[4] https://github.com/Piletskii-Oleg/UnityGame (дата обращения: 15.12.2022)

Список литературы

[1] Ryan Hipple. Game Architecture with Scriptable Objects. — URL: https://www.youtube.com/watch?v=raQ3iHhE_Kk (online; accessed: 2022-12-13).

[2] Unity Documentation. Coroutine. — URL: https://docs.unity3d.com/Manual/Coroutines.html (online; accessed: 2022-12-13).

[3] Unity Documentation. MonoBehaviour. — URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html (online; accessed: 2022-12-13).

[4] Unity Documentation. Physics.SphereCast. — URL: https://docs.unity3d.com/ScriptReference/Physics.SphereCast.html (online; accessed: 2022-12-13).

[5] Unity Documentation. ScriptableObject. — URL: https://docs.unity3d.com/Manual/class-ScriptableObject.html (online; accessed: 2022-12-04).

[6] Unity Technologies. Adventure - Sample Game. — URL: https://assetstore.unity.com/packages/templates/tutorials/adventure-sample-game-76216 (online; accessed: 2022-12-13).